Minimal Synchrony for Asynchronous Byzantine Consensus

Zohir Bouzid, Achour Mostefaoui, Michel Raynal

Publications Internes de l'IRISA
ISSN : 2102-6327
PI 2025 – January 2015

Minimal Synchrony for Asynchronous Byzantine Consensus

Zohir Bouzid* Achour Mostéfaoui** Michel Raynal*** ****

Abstract: Solving the consensus problem requires in one way or another that the underlying system satisfies some synchrony assumption. Considering an asynchronous message-passing system of n processes where (a) up to $t < n/3$ may commit Byzantine failures, and (b) each pair of processes is connected by two uni-directional channels (with possibly different timing properties), this paper investigates the synchrony assumption required to solve consensus, and presents a signature-free consensus algorithm whose synchrony requirement is the existence of a process that is an *eventual* $\langle t+1 \rangle$*bisource*. Such a process p is a correct process that eventually has (a) timely input channels from t correct processes and (b) timely output channels to t correct processes (these input and output channels can connect p to different subsets of processes). As this synchrony condition was shown to be necessary and sufficient in the stronger asynchronous system model (a) enriched with message authentication, and (b) where the channels are bidirectional and have the same timing properties in both directions, it follows that it is also necessary and sufficient in the weaker system model considered in the paper. In addition to the fact that it closes a long-lasting problem related to Byzantine agreement, a noteworthy feature of the proposed algorithm lies in its design simplicity, which is a first-class property.

Key-words: Adopt-commit, Asynchronous message-passing, Byzantine process, Consensus, Distributed algorithm, Eventual timely channel, Feasibility condition, Lower bound, Optimal resilience, Reliable broadcast, Signature-free algorithm, Synchrony assumption.

Consensus Byzantin avec Synchronisme Minimal

Résumé : *Cet rapport présente le premier algorithme qui résout le consensus en dépit de processus byzantins, et cela avec des hypothèses de synchronisme minimales.*

Mots clés : *Système à passage de messages, Processus byzantin, Synchronie minimale.*

---

* IRISA-ASAP : équipe commune avec l'Université de Rennes 1 et Inria
** LINA, Université de Nantes, 44322 Nantes Cedex, France
*** Institut Universitaire de France
**** IRISA-ASAP : équipe commune avec l'Université de Rennes 1 et Inria

1 Introduction

Byzantine consensus A process has a *Byzantine* behavior when it behaves arbitrarily [25]. This bad functioning can be intentional (malicious behavior, e.g., due to intrusion) or simply the result of a transient fault that altered the local state of a process, thereby modifying its execution in an unpredictable way.

We are interested here in the *consensus* problem in message-passing distributed systems prone to Byzantine process failures whatever their origin. Consensus is an agreement problem in which each process first proposes a value and then decides on a value [25]. In a Byzantine failure context, the consensus problem is defined by the following properties: every non-faulty process decides (termination), no two non-faulty processes decide differently (agreement), and the decided value is not arbitrary, i.e., it is related in one way or another to values proposed by non-faulty processes (validity).

Context of the paper A synchronous distributed system is characterized by the fact that both processes and communication channels are synchronous (or timely) [3, 20, 27]. This means that there are known bounds on process speed and message transfer delays. Let t denote the maximum number of processes that can be faulty in a system made up of n processes. In a synchronous system, consensus can be solved (a) for any value of t (i.e., $t < n$) in the crash failure model, (b) for $t < n/2$ in the general omission failure model, and (c) for $t < n/3$ in the Byzantine failure model [19, 25]. Moreover, these bounds are tight.

Differently, when all channels are asynchronous (i.e., when there is no bound on message transfer delays), it is impossible to solve consensus even if we consider the weakest failure model (namely, the process crash failure model) and assume that at most one process may be faulty (i.e., $t = 1$) [14]. It trivially follows that Byzantine consensus is impossible to solve in a failure-prone asynchronous distributed system.

As Byzantine consensus can be solved in a synchronous system and cannot in an asynchronous system, a natural question that comes to mind is the following "*When considering the synchrony-to-asynchrony axis, which is the weakest synchrony assumption that allows Byzantine consensus to be solved in a message-passing system?*" This long-lasting question is the issue addressed in this paper. To that end, the paper considers a synchrony assumption capturing the structure and the number of eventually synchronous channels among correct processes.

Related work Several approaches to solve Byzantine consensus have been proposed. We consider here only deterministic approaches[1]. One consists in enriching the asynchronous system (hence the system is no longer fully asynchronous) with a failure detector, namely, a device that provides processes with (possibly unreliable) hints on failures [9]. Basically, in one way or another, a failure detector encapsulates synchrony assumptions. Failure detectors suited to Byzantine behavior have been proposed and used to solved Byzantine consensus (e.g., [12, 15, 18]).

Another approach proposed to solve Byzantine consensus consists in directly assuming that some channels satisfy a synchrony property ("directly" means that the synchrony property is not hidden inside a higher level abstraction such as a failure detector). This approach, which relies on the notion of an $\Diamond\langle x+1\rangle$*bisource* (read "$\Diamond$" as "eventual"), was introduced in [1]. Intuitively, this notion states that there is a correct process that has x input channels from correct processes and x output channels to correct processes that are eventually timely [11, 13] (the "+1" comes from the fact that it is assumed that each process has a "virtual" input/output channel from itself to itself, which is always timely).

Considering asynchronous systems with Byzantine processes without message authentication, it is shown in [1] that Byzantine consensus can be solved if the system has an $\Diamond\langle n-t\rangle$bisource (all other channels being possibly fully asynchronous). Moreover, the process that is the $\Diamond\langle n-t\rangle$bisource can never be explicitly known by the whole set of processes. Considering systems with message authentication, a Byzantine consensus algorithm is presented in [24] that requires an $\Diamond\langle t+1\rangle$bisource only. As for Byzantine consensus in synchronous systems, all these algorithms assume $t < n/3$. Finally, it has been shown in [4] that the "$\Diamond\langle t+1\rangle$bisource" synchrony assumption is a necessary and sufficient condition to solve Byzantine consensus in asynchronous bi-directionnal message-passing systems, enriched with message authentication.

[1]Enriching the system with random numbers allows for the design of randomized Byzantine consensus algorithms. These algorithms are characterized by a probabilistic termination property (e.g., [5, 8, 21, 26]).

Content of the paper This paper presents a Byzantine consensus algorithm for signature-free asynchronous message-passing systems, which requires only the two assumptions: $t < n/3$ and the existence of an $\diamond\langle t+1\rangle$bisource. As these assumptions are necessary and sufficient to solve Byzantine consensus in the asynchronous model enriched with message authentication [4], it follows that (a) the existence of an $\diamond\langle t+1\rangle$bisource is necessary and sufficient to solve Byzantine consensus in an asynchronous signature-free system, and (b) the proposed algorithm is optimal with respect to underlying synchrony assumptions.

The proposed algorithm is round-based. To attain its goal, it relies on a modular construction involving two communication abstractions and two distributed objects. More precisely, we have the following.

- The communication abstractions are the one-to-all *reliable broadcast* (RB) abstraction introduced in [6], and a very simple new communication abstraction that we call *cooperative broadcast* (CB). As suggested by its name, it is an all-to-all broadcast abstraction. This abstraction, which uses RB as an underlying subroutine, is particularly simple. It actually captures important cooperation properties, which make easier the design of upper layer distributed agreements.
- The two distributed objects are the following ones (the implementation of each of them use the underlying CB broadcast abstraction).
  - The first object is a message-passing version of the *adopt-commit* (AC) object (introduced in [16]) appropriately modified to cope with up to $t < n/3$ Byzantine processes. Each round of the consensus algorithm uses a specific AC object. The aim of these objects is to prevent the consensus safety property from being violated.
  - The second object is a round-based object called *eventual agreement* (EA) object. It aims is to ensure the consensus termination property. Hence, its implementation relies on the $\diamond\langle t+1\rangle$bisource assumption.

It is important to emphasize that, when designing the algorithm presented in the paper, modularity and simplicity were considered as first class design criteria. The algorithm presented is only the last step of a long quest: “Simplicity does not precede complexity, but follows it” (Alan Perlis, First Turing Award).

Road map The paper is made up of 7 sections. Section 2 presents the basic underlying asynchronous Byzantine computation model, the RB broadcast abstraction and the new CB broadcast abstraction. Section 3 presents an AC object suited to message-passing systems prone to Byzantine failures. Then, Section 4 presents the $\diamond\langle t+1\rangle$bisource behavioral assumption. Section 5 presents the round-based eventual agreement object. Section 6 pieces together the previous abstractions to obtain the synchrony-optimal Byzantine consensus algorithm. Finally, Section 7 concludes the paper.

2 Basic Model, Reliable Broadcast, and Cooperative Broadcast

2.1 Processes, communication network, and failure model

Asynchronous processes The system is made up of a finite set Π of $n > 1$ sequential processes, namely $\Pi = \{p_1, \ldots, p_n\}$. As local processing times are negligible with respect to message transfer delays, they are considered as being equal to zero. Both notations $i \in Y$ and $p_i \in Y$ are used to say that p_i belongs to the set Y.

Communication network The processes communicate by exchanging messages through an asynchronous reliable point-to-point network. “Asynchronous” means that there is no bound on message transfer delays. “Reliable” means that the network does not loss, duplicate, modify, or create messages. “Point-to-point” means that any pair of processes is connected by two uni-directional channels (one in each direction). Hence, when a process receives a message, it can identify its sender. Moreover, as there is no message loss, all message transfer delays are finite.

A process p_i sends a message to a process p_j by invoking the primitive "send $\text{TAG}(m)$ to p_j", where TAG is the type of the message and m its content. To simplify the presentation, it is assumed that a process can send messages to itself. A process receives p_i a message by executing the primitive "receive()". Then then say that the message is *received* by p_i.

Failure model Up to t processes can exhibit a *Byzantine* behavior. A Byzantine process is a process that behaves arbitrarily: it can crash, fail to send or receive messages, send arbitrary messages, start in an arbitrary state, perform arbitrary state transitions, etc. Moreover, Byzantine processes can collude to "pollute" the computation (e.g., by sending messages with the same content, while they should send messages with distinct content if they were non-faulty).

A process that exhibits a Byzantine behavior is called *faulty*. Otherwise, it is *correct* or *non-faulty*. Given an execution, $\mathcal{C}$ denotes the set of processes that are correct in this execution.

Let us notice that, as each pair of processes is connected by a channel, no Byzantine process can impersonate another process. Moreover, it is assumed that the Byzantine processes do not control the network (they can neither corrupt the messages sent by non-faulty processes, nor modify the message reception schedule).

Discarding messages from Byzantine processes If, according to its algorithm, a process p_j is assumed to send a single message $\text{TAG}()$ to a process p_i, then p_i processes only the first message $\text{TAG}(v)$ it receives from p_j. This means that, if p_j is Byzantine and sends several messages $\text{TAG}(v)$, $\text{TAG}(v')$ where $v' \neq v$, etc., all of them except the first one are discarded.

Unreliable (best effort) broadcast This simple broadcast is defined by a pair of operations denoted broadcast() and receive(), where broadcast $\text{TAG}(m)$ is used as a shortcut for

for each $j \in \{1, \ldots, n\}$ send $\text{TAG}(m)$ to p_j **end for**.

This means that a message *broadcast* by a correct process is *received* at least by all the correct processes. Differently, while it is assumed to send the same message to all the processes, a faulty process can actually send different messages to distinct processes and no message to others.

Notation The notation $\mathcal{BZ\_AS}_{n,t}[\emptyset]$ is used to denote the previous basic Byzantine asynchronous message-passing computation model.

2.2 Reliable broadcast abstraction

This broadcast abstraction (in short, RB-broadcast) was proposed by G. Bracha [6]. It is a one-shot one-to-all communication abstraction, which provides processes with two operations denoted RB_broadcast() and RB_deliver(). When p_i invokes RB_broadcast() (resp., RB_deliver()), we say that it "RB-broadcasts" a message (resp., "RB-delivers" a message). An RB-broadcast instance where process p_x is the sender is defined by the following properties.

- RB-Validity. If a non-faulty process RB-delivers a message m (from p_x), then, if p_x is correct, it RB-broadcast m.
- RB-Unicity. A correct process RB-delivers at most one message from p_x.
- RB-Termination-1. If p_x is non-faulty and RB-broadcasts a message m, all the non-faulty processes eventually RB-deliver m from p_x.
- RB-Termination-2. If a non-faulty process RB-delivers a message m from p_x (possibly faulty) then all the non-faulty processes eventually RB-deliver the same message m from p_x.

The RB-Validity property relates the output to the input, while RB-Unicity states that there is no message duplication. The termination properties state the cases where processes have to RB-deliver messages. The second of them is what makes the broadcast reliable. It is shown in [7] that $t < n/3$ is an upper bound on t when one has to implement such an abstraction. For self-containment of the paper, an algorithm implementing RB-broadcast is described in Appendix B (this algorithm is from [6]).

Notation The basic computing model strengthened with the additional constraint $t < n/3$ is denoted $\mathcal{BZ\_AS}_{n,t}[t < n/3]$. RB-broadcast can consequently be implemented in this model.

2.3 Cooperative broadcast abstraction

Definition This new communication abstraction (in short CB-broadcast) is a one-shot all-to-all broadcast defined by an operation, denoted $\mathsf{CB\_broadcast}()$, plus a read-only per process p_i, denoted cb_valid_i. "All-to-all" means that it is assumed that all correct processes invoke $\mathsf{CB\_broadcast}()$. When a process p_i invokes $\mathsf{CB\_broadcast}(v)$, we say that "it cb-broadcasts v".

An invocation of $\mathsf{CB\_broadcast}()$ by a process p_i has an input parameter, namely the value that p_i wants to broadcast, and returns a value, which is a value CB-broadcast by a correct process. The CB-broadcast abstraction is formally defined by the following properties.

- CB-Operation Termination. The invocation of $\mathsf{CB\_broadcast}()$ by a correct process terminates.
- CB-Operation Validity. If the invocation of $\mathsf{CB\_broadcast}()$ returns v to a correct process p_i, $v \in cb\_valid_i$.
- CB-Set Termination. The set cb_valid_i of a correct process p_i is eventually non-empty.
- CB-Set Validity. The set cb_valid_i of any correct process p_i contains only values cb-broadcast by correct processes.
- CB-Set Agreement. The set cb_valid_i and cb_valid_j of any two correct processes p_i and p_j are eventually equal.

Feasibility condition in the presence of up to t Byzantine processes Let m be the number of different values that can be cb-broadcast by correct processes. It follows from the previous specification that, even when the (at most) t Byzantine processes propose a same value w, not proposed by a correct process, w can neither be returned, nor belong to the set cb_valid_i of a correct process p_i. This can be ensured if and only if there is a value cb-broadcast by at least $(t+1)$ correct processes. This feasibility condition is captured by the predicate $n - t > mt$. (A proof of this feasibility condition can be found in [17]).

Hence, we assume in the following that at most $m \leq \lfloor \frac{n-(t+1)}{t} \rfloor$ different values can be cb-broadcast by the set of correct processes, and the corresponding abstraction is called an *m-valued* CB-broadcast.

An algorithm implementing CB-broadcast A simple algorithm implementing CB-broadcast is described in Figure 1. When p_i invokes $\mathsf{CB\_broadcast}(v_i)$, it first invokes $\mathsf{RB\_broadcast}$ $\text{CB\_VAL}(v_i)$ for all correct processes to be eventually aware of v_i (line 1). Then, it waits until its set cb_valid_i becomes non-empty (line 2). When this occurs, p_i takes any value from cb_valid_i and returns it (line 3). Finally, p_i adds to cb_valid_i all the values it RB-delivers from $(t+1)$ different processes (i.e., v was RB-broadcast by at least one correct process). It is important to notice that, after the predicate $cb\_valid_i \neq \emptyset$ became satisfied, new values can still be added to cb_valid_i.

```
operation CB_broadcast(v_i) is
(1)   RB_broadcast CB_VAL(v_i);
(2)   wait (cb_valid_i ≠ ∅);
(3)   return (any value in cb_valid_i).

when CB_VAL(v) is RB_delivered from p_j do
(4)   if (v RB_delivered from (t + 1) diff. processes) then add v to cb_valid_i end if.
```

Figure 1: An algorithm implementing m-valued CB-broadcast in $\mathcal{BZ\_AS}_{n,t}[t < n/3]$

Theorem 1. *The algorithm described in Figure 1 implements the m-valued CB-broadcast abstraction in $\mathcal{BZ\_AS}_{n,t}[t < n/3]$.*

Proof Proof of the CB-Termination properties.
It follows from the feasibility condition, that there is a value v that is proposed by at least $(t+1)$ correct processes. Hence, these processes RB-broadcast $\text{CB\_VAL}(v)$. It then follows from line 4 and the RB-termination property that v will be added to the set cb_valid_i of each correct process p_i. Hence, the CB-Set Termination property is satisfied, and no correct process can be blocked forever at line 2, from which follows the CB-Operation Termination property.

Proof of the CB-Validity properties.
To prove the CB-Set Validity property, let us consider a value v cb-broadcast only by Byzantine processes. It follows that a correct process p_i can RB-deliver v from at most t different processes. Hence, p_i cannot add v to cb_valid_i at line 4, which proves the property. The CB-Operation Validity property is then a trivial consequence of the CB-Set Validity property.

Proof of the CB-Set Agreement property.
Let us consider a value $v \in cb\_valid_i$. This means that p_i RB-delivered the message $\text{CB\_VAL}(v)$ from $(t+1)$ different processes (line 4). It then follows from the RB-termination property of RB-broadcast that each correct process p_j RB-delivers these $(t+1)$ messages $\text{CB\_VAL}(v)$. Consequently, any correct process p_j adds v to its local set cb_valid_j, which concludes the proof. $\Box_{Theorem\ 1}$

3 Adopt-Commit in the Presence of Byzantine Processes

This object was introduced in [16] in the context of read/write communication. Here we slightly modify its definition to cope with Byzantine processes (which by definition can decide anything).

Definition An adopt-commit (AC) is a one-shot object which encapsulates the safety part of agreement problems. It provides processes with a single operation denoted AC_propose(). This operation takes a value as input parameter (we say that the invoking process proposes this value), and returns a pair $\langle d, v\rangle$ (we say that the invoking process decides $\langle d, v\rangle$), where d is a control tag and v a value. An AC object is defined by the following properties.

- AC-Termination. An invocation of AC_propose() by a correct process terminates.
- AC-Validity. This property is made up of two parts.
  - AC-Output domain. If a correct process decides $\langle d, v\rangle$, $d \in \{\texttt{commit}, \texttt{adopt}\}$, and v is a value that was proposed by a correct process.
  - AC-Obligation. If all the correct processes propose the same value v, only the pair $\langle \texttt{commit}, v\rangle$, can be decided.
- AC-Quasi-agreement. If a correct process decides $\langle \texttt{commit}, v\rangle$, no other correct process can decide $\langle -, v'\rangle$ where $v' \neq v$.

Implementations of an AC object in the presence of process crash failures can be found in [16, 22]. The implementation of [16] is for asynchronous systems where any number of processes may crash and communication is by atomic read/write registers. The implementation of [22] is for asynchronous message-passing systems where a minority of processes may crash.

It follows from the AC-Output domain property, that a value proposed only by Byzantine processes cannot be decided by a correct process. This means that an AC object has the same feasibility condition as CB-broadcast (let us also notice that this is independent from the fact that an AC object can be built on top of CB-broadcast). Hence, we assume that at most $m \leq \lfloor \frac{n-(t+1)}{t} \rfloor$ values can be proposed by the correct processes, and the corresponding object is called an *m-valued* adopt-commit object.

Implementation of an m-valued adopt-commit object A distributed algorithm implementing an AC object in the presence of up to $t < n/3$ Byzantine processes is described in Figure 2, for a correct process p_i. This algorithm is based on an underlying CB-broadcast, which means that each process has a read-only local set cb_val_i (initially empty).

operation AC_propose(v_i) **is**
(1) $est_i \leftarrow$ CB_broadcast AC_PROP(v_i);
(2) RB_broadcast AC_EST(est_i);
(3) wait (AC_EST(est) messages have been RB-delivered from $(n-t)$
different processes, and their est values belong to cb_valid_i);
(4) $mfv_i \leftarrow$ most frequent value in the previous $(n-t)$ AC_EST() messages;
(5) **if** (each of the previous $(n-t)$ AC_EST() messages carries mfv_i)
(6) **then** return ($\langle \texttt{commit}, mfv_i \rangle$)
(7) **else** return ($\langle \texttt{adopt}, mfv_i \rangle$)
(8) **end if**.

Figure 2: An algorithm implementing an m-valued adopt-commit object in $\mathcal{BZ\_AS}_{n,t}[t < n/3]$

When a process p_i invokes AC_propose(v_i), it first issues CB_broadcast AC_PROP(v_i) from which it obtains a value that it saves in est_i (line 1). It then RB-broadcasts the message AC_EST(est_i) (line 2), and waits until (a) it has RB-delivered messages AC_EST() from $(n-t)$ different processes, and (b) the values carried by these messages belong to the set cb_valid_i supplied by CB-broadcast (line 3). Let us remember that, after the predicate $cb\_valid_i \neq \emptyset$ became satisfied, new values can still be added to cb_valid_i.

When this predicate becomes satisfied, p_i computes the most frequent value mfv_i carried by the previous $(n-t)$ AC_EST() messages (line 4). If there are several "most frequent" values, p_i takes any of them. Finally, if all the messages who made satisfied the predicate of line 3 carried the same value mfv_i, p_i returns the pair $\langle \texttt{commit}, mfv_i \rangle$) (line 6); otherwise it returns the pair $\langle \texttt{adopt}, mfv_i \rangle$) (line 7).

Theorem 2. *Assuming that all the correct processes invoke* AC_propose()*, the algorithm described in Figure* 2 *implements an* m-valued adopt-commit *object in* $\mathcal{BZ\_AS}_{n,t}[t < n/3]$.

Proof Proof of the AC-termination property.
Due to the CB-Operation termination property, no correct process blocks forever at line 1. So, we have only toshow that no correct process can block forever at line 3. It follows from CB-Set Termination and CB-Set Validity that the sets cb_valid_i of the correct processes are eventually not empty and contain only values proposed by correct processes. As (i) the value RB-broadcast by each correct process at line 2 is a value of its set cb_valid_i, (ii) there are at least $(n-t)$ correct processes, and (iii) the sets cb_valid_i of the correct processes are eventually equal (CB-Set Agreement property), it follows that the predicate of line 3 is eventually satisfied at each correct process, which concludes the proof of AC-termination property.

Proof of the AC-Output domain property.
Let us first observe that a correct process can decide only $\langle \texttt{commit}, v \rangle$ or $\langle \texttt{adopt}, v \rangle$ (lines 6-7). Hence, we have only to show that v is a value proposed by a correct process. A value v decided by a correct process p_i was RB-delivered in a message AC_EST(v). It follows from the predicate of line 3 that $v \in cb\_valid_i$. Finally, if follows from the CB-Set Validity property that v is a value proposed by a correct process.

Proof of the AC-Obligation property.
If all correct processes propose the same value v, it follows from the CB-Set (Termination, Validity, and Agreement) properties that the set cb_valid_i of each correct process p_i is eventually equal to $\{v\}$. Hence, each correct process RB-broadcasts the message AC_EST(v) at line 2. It then follows from the predicate of line 3 that no value different from v can be decided.

Proof of the AC-Quasi-agreement property.
Let p_i and p_j be two correct processes such that p_i decides $\langle \texttt{commit}, v\rangle$ while p_j decides $\langle -, v'\rangle$. As p_i decides $\langle \texttt{commit}, v\rangle$, it follows from line 3 that it RB-delivered the message $\text{AC\_EST}(v)$ from $(n-t)$ different processes. As, due to the RB-Unicity and RB-Termination-2 properties, no two correct processes RB-deliver different values from the same process, it follows that, among the $(n-t)$ messages $\text{AC\_EST}()$ RB-delivered by p_j, at most t of them may carry a value different from v, i.e., at least $n - 2t \geq t + 1$ carry the value v. It follows that v is the most frequent value RB-delivered by p_j, and consequently $v' = v$. $\Box_{Theorem\ 2}$

4 The $\Diamond\langle t+1\rangle$Bisource Assumption

4.1 Definitions

These notions were introduced in [1, 13] (see also Appendix A).

Eventually timely channel Let us consider the channel connecting a process p_i to a process p_j. This channel is *eventually timely* if there is a finite time τ and a bound δ, such that any message sent by p_i to p_j at time τ' is received by p_j by time $\max(\tau, \tau') + \delta$. Let us observe that neither τ, nor δ, is known by the processes.

As already indicated, there is an input/output channel from each process to itself.

$\Diamond[k]$sink, $\Diamond[k]$source, and $\Diamond[k]$bisource A correct process p_i is an $\Diamond[k]$sink if it has eventually timely input channels from k correct processes (including itself). This set of processes is denoted X_i^-. Similarly, a correct process is an $\Diamond[k]$source if it has k eventually timely output channels to correct processes (including itself). This set of processes is denoted X_i^+.

An $\Diamond[k]$bisource is a correct process p_i that is both $\Diamond[k]$sink and $\Diamond[k]$source. Let us remark that the timely input channels and the timely output channels do not necessarily connect p_i to the same subset of processes.

Notation for system models The system model $\mathcal{BZ\_AS}_{n,t}[t < n/3]$ enriched with an $\Diamond\langle t+1\rangle$bisource is denoted $\mathcal{BZ\_AS}_{n,t}[t < n/3, \Diamond\langle t+1\rangle\text{bisource}]$.

5 Eventual Agreement Object

5.1 Motivation and definition

This object, which is round-based, will be used to ensure the termination of the consensus algorithm, namely, its aim is to allow the correct processes to eventually converge on a single value. To this end, it provides the processes with a single operation denoted $\mathsf{EA\_propose}(r, v)$ where r is a round number and v is the value proposed at this round by the invoking process. Each invocation of $\mathsf{EA\_propose}()$ by a correct process returns a value. It is assumed that each correct process invokes this operation once per round, and its successive invocations are done according to consecutive round numbers. When a process invokes $\mathsf{EA\_propose}(r, v)$, we say that it "ea-proposes v at round r".

Definition An eventual agreement (EA) object is defined by the following properties.

- EA-Termination. For any r, if all correct processes invoke $\mathsf{EA\_propose}(r, -)$, each of these invocations terminates.
- EA-Validity. For any r, if all correct processes invoke $\mathsf{EA\_propose}(r, v)$ no correct process returns a value different from v.

- EA-Eventual agreement. If the correct processes execute an infinite number of rounds, there is an infinite number of rounds r at which all the correct processes return the same value v, where v is such that a correct process invoked $\mathsf{EA\_propose}(r, v)$.

It is important to notice that the EA-Validity property is particularly weak. More precisely, if, during a round r, two correct processes invoke $\mathsf{ea\_propose}(r, v1)$ and $\mathsf{EA\_propose}(r, v2)$, with $v1 \neq v2$, the invocation of $\mathsf{EA\_propose}(r, -)$ by any correct process is allowed to return an arbitrary value (i.e., even a value proposed neither by a correct nor by a Byzantine process).

As the implementation that follows uses at every round an instance of CB-broadcast, we assume that at most $m \leq \lfloor \frac{n-(t+1)}{t} \rfloor$ different values are ea-proposed by correct processes.

5.2 Implementation of an m-valued eventual agreement object

Definitions The algorithm presented below uses the following sets and functions.

- There are $\alpha = \binom{n}{n-t}$ possible combinations of $(n-t)$ processes among the n processes p_1, ..., p_n. Let us call them $F_1 \ldots F_\alpha$.
- Given any round number $r \geq 1$:
  - $\mathsf{coord}(r)$ denotes the function $\big((r-1) \bmod \mathsf{n}\big) + 1$.
    Given a round r, $\mathsf{coord}(r)$ defines its coordinator process. As we can see, if there is an infinite number of rounds, each process is infinitely often round coordinator.
  - $F(r)$ denotes the function $F_{\mathsf{index}(r)}$, where $\mathsf{index}(r) = \big((\lceil \frac{r}{n} \rceil - 1) \bmod \alpha\big) + 1$.
    Hence, each set $F(r)$ returns a set made up of $(n-t)$ processes. During each round, its coordinator strives to decide a value. To this end, it requires the help of the processes in $F(r)$ to broadcast the value it champions. F_1 is used by the coordinators of the rounds 1 to n; F_2 is used by the coordinators of the rounds $(n+1)$ to $2n$; ..., F_α is used by the coordinators of the rounds $((\alpha-1)n+1)$ to αn; F_1 is used by the coordinators of the rounds $((\alpha n+1)$ to $(\alpha+1)n$; etc.

Considering an infinite sequence of rounds, it is important to notice that there is an infinite number of rounds r and r' such that $(\mathsf{coord}(r) = \mathsf{coord}(r')) \wedge F(r) = F(r')$ and an infinite number of rounds r and r' such that $\big(\mathsf{coord}(r) = \mathsf{coord}(r')\big) \wedge \big(F(r) \neq F(r')\big)$.

Local variables Each process p_i manages the following local variables.

- $timer_i[1..]$ is an array of timers, such that $timer_i[r]$ is the timer used by p_i for round r.
- $CB[1..]$ is an array of CB-broadcast instances shared by all processes. $CB[r]$ is the instance associated with round r. Hence, $CB[r].cb\_valid_i$ is the set of values supplied to p_i by $CB[r]$.

To distinguish messages which have the same tag but are sent at different rounds, a message XXX() associated with round r is denoted XXX$[r]()$.

Algorithm: first part of $\mathsf{EA\_propose}()$ (Lines 1-5) The algorithm executed by a correct process p_i is described in Figure 3. Let us remind that, it is assumed that each correct process invokes $\mathsf{EA\_propose}()$ at every round.

When a correct process p_i invokes $\mathsf{EA\_propose}(r_i, val_i)$ (r_i is a round number and val_i the value it ea-proposes at this round), it first invokes $CB[r_i].\mathsf{CB\_broadcast}$ EA_PROP1(val_i), and saves the value returned in aux_i (line 1).Then, p_i broadcasts the message EA_PROP2$[r_i](aux_i)$ (line 1) and waits until (a) it has received messages EA_PROP2$[r_i]()$ from $(n-t)$ different processes, and (b) the values carried by these messages belong to the set $CB[r_i].cb\_valid_i$ supplied by

operation EA_propose(r_i, val_i) **is**
(1) $aux_i \leftarrow CB[r_i]$.CB_broadcast EA_PROP1(val_i);
(2) broadcast EA_PROP2$[r_i](aux_i)$;
(3) wait (EA_PROP2$[r_i]()$ messages have been received from $(n-t)$
different processes, and their aux values belong to $CB[r_i].cb\_valid_i$);
(4) **if** (the $(n-t)$ previous messages carry the same value v) **then** return(v) **end if**;
(5) set $timer_i[r_i]$ to r_i;
(6) wait (EA_RELAY$[r_i](aux)$ messages received from $(n-t)$ different processes);
(7) **if** (EA_RELAY$[r_i](v)$ where $v \neq \perp$ received from a process in $F(r_i)$)
(8) **then** return(v)
(9) **else** return(val_i)
(10) **end if**.

when EA_PROP2$[r]()$ **is received from** a process in $F(r)$ **do**
(11) **if** $\big((i = \mathsf{coord}(r)$ $\wedge$ (EA_COORD$[r]()$ not already broadcast)$\big)$
(12) **then let** w be the value carried by the message EA_PROP2$[r]()$;
(13) broadcast EA_COORD$[r](w)$
(14) **end if**.

when EA_COORD$[r](v)$ **is received from** $p_{\mathsf{coord}(r)}$ or ($timer_i[r]$ expires) **do**
(15) **if** (EA_RELAY$[r]()$ not already broadcast)
(16) disable $timer_i[r]$;
(17) **if** ($timer_i[r]$ expired) **then** $v\_coord_i \leftarrow \perp$ **else** $v\_coord_i \leftarrow v$ **end if**;
(18) broadcast EA_RELAY$[r](v_coord_i)$
(19) **end if**.

Figure 3: An algorithm implementing an m-valued EA object in $\mathcal{BZ\_AS}_{n,t}[t < n/3, \diamond\langle t+1\rangle\text{bisource}]$

CB-broadcast instance $CB[r_i]$ (line 3). If all these messages carry the same value v, p_i returns v as result of its invocation EA_propose(r_i, val_i) (line 4)[2]. Otherwise, p_i sets the timer associated with the round r_i to the value r_i (line 5)[3].

Algorithm: message processing and role of the round coordinator (Lines 11-19) Each round r uses a round coordinator, defined by coord(r). As we have also seen, the set of $(n-t)$ processes denoted $F(r)$ is associated with round r.

When p_i is the coordinator of round r and receives for the first time a message EA_PROP2$[r]()$ from a process in the set $F(r)$, it champions the value w carried by this message to become the value returned by the invocations of EA_propose$(r, -)$. To that end, it simply broadcasts the message EA_COORD$[r](w)$ (lines 11-14).

When a process p_i receives a message EA_COORD$[r](v)$ from the coordinator of round r, if not yet done, it broadcasts the message EA_RELAY$[r](v)$ to inform the other processes that it has received the value v championed by the coordinator of round r. If the local timer associated with this round ($timer_i[r]$) has already expired, p_i broadcasts the message EA_RELAY$[r](\perp)$, to inform the other processes that it suspects the coordinator of round r not to be an $\diamond\langle t+1\rangle$bisource (this suspicion can be due to the asynchrony of the channel connecting $p_{\mathsf{coord}(r)}$ to p_i, or the fact that –while $p_{\mathsf{coord}(r)}$ is an $\diamond\langle t+1\rangle$bisource– the link from $p_{\mathsf{coord}(r)}$ to p_i is not yet synchronous, or the fact that $p_{\mathsf{coord}(r)}$ has a Byzantine behavior). In all cases, as $timer_i[r]$ will no longer be useful, p_i disables it. This behavior of p_i is captured by the lines 15-19.

[2]Let us remark that lines 1-3 of Figure 3 and lines 1-3 of Figure 2 differ only in the fact that an RB-broadcast is used at line 2 for the AC object, and a simple broadcast is used at line 2 for the EA object. These lines have not been encapsulated to define a higher level object because the messages EA_PROP2$[r_i]()$ are explicitly used in lines 11-14 of Figure 3, while their counterparts in an AC object –messages AC_EST()– are not used by the upper layer.

[3]The important point here is that the value of the timer increases; as r_i increases at every round, it is used as a timeout value. More generally, it is possible to assign to $timer_i[r_i]$ the value returned by an increasing function $f_i(r_i)$, which can be specific to each process p_i.

Algorithm: second part of EA_propose() (Lines 6-10) After it has set $timer_i[r_i]$ (line 5), p_i waits until it has received a message EA_RELAY$[r_i]()$ from $(n-t)$ different processes (line 6). When this occurs, the invocation of EA_propose(r_i, val_i) by p_i returns a value. This value is $v \neq \perp$ if p_i received a message EA_RELAY$[r_i](v)$ from a process in the set $F(r_i)$ (lines 7-8). Otherwise, no process of $F(r_i)$ witnesses the value championed by the coordinator of round r. In this case, p_i returns the value val_i, i.e., the value it ea-proposed (line-9).

5.3 Proof

Let us remember that, by assumption, all correct processes invoke EA_propose$(r, -)$, where $r = 1$. Moreover, they ea-propose at most m different values.

Lemma 1. *For any r, if all correct processes invoke* EA_propose(r, v) *no correct process returns a value different from* v.

Proof Since all correct processes invoke EA_propose(r, v), all of them invoke $CB[r]$.CB_broadcast EA_PROP1(v) at line 1. It follows from the CB-Set Validity property that the set $CB[r].cb\_valid_i$ of any correct process p_i can contain only the value v. Therefore, the $(n-t)$ messages EA_PROP$[r]()$ considered at line 3 carry necessarily the value v. Consequently, every correct process that executes line 4 necessarily returns v. That is, no other value can be returned by a correct process. $\Box_{Lemma\ 1}$

Lemma 2. *Let $r \geq 1$. If all correct processes invoke* EA_propose$(r, -)$*, then each of these invocation terminates.*

Proof Let p_i a correct process. Let us first observe that, due to the CB-Operation Termination property, p_i cannot be blocked forever when it invokes $CB[r]$.CB_broadcast EA_PROP1() at line 1. Hence, the proof consists in showing that p_i cannot block forever at line 3 or line 6.

The proof that p_i cannot block forever at line 3 follows from the following observation. As a correct process cannot block forever at line 1, each correct process p_j broadcasts the message EA_PROP2$[r](aux_j)$ at line 2 where aux_j was returned by its invocation of $CB[r]$.CB_broadcastEA_PROP1() at line 1. It follows from the CB-Operation Validity property that $aux_j \in CB[r].cb\_valid_j$. Moreover, according to the CB-Set Agreement property, aux_j is eventually added to the set $CB[r].cb\_valid_i$ of every correct process p_i. Consequently, each correct process p_i receives messages EA_PROP2$[r]()$ from at least $(n-t)$ processes and the values carried by these messages eventually belong to its set $CB[r].cb\_valid_i$. Consequently no correct process remains blocked forever at line 3.

It remains to prove that p_i cannot block forever at line 6. Let us observe that since $timer_i[r]$ is set to r, it follows that this timer expires at every correct process. Hence, the when condition preceding line 15 becomes true at every correct process and each of them broadcasts an EA_RELAY$[r]()$ message if not already done. It follows that p_i receives messages EA_RELAY$[r]()$ from at least $(n-t)$ processes. Therefore, p_i cannot be blocked forever waiting at line 6. $\Box_{Lemma\ 2}$

Lemma 3. *If the correct processes execute an infinite number of rounds, there is an infinite number of rounds r at which all the correct processes return the same value v, where v is such that a correct process invoked* EA_propose(r, v).

Proof Let us define the following rounds:

- Let r_1 be the first round that is strictly greater than 2δ.
- Let p_ℓ be an $\Diamond[t+1]$bisource. There exists a round r_2 such that in every subsequent round:
  - Each message sent by any $p_x \in X_\ell^-$ to p_ℓ is received within an interval of at most δ time units.
  - Each message sent by p_ℓ to any $p_y \in X_\ell^+$ is received within an interval of at most δ time units.
- Let $r > \max(r_1, r_2)$ be any round coordinated by p_ℓ such that $X_\ell^+ \subseteq F(r)$ and $F(r) \subseteq \mathcal{C}$. Let us notice that, due to the definition of $F(r)$, an infinity of such rounds r exists.

Claim C. For every process $p_i \in X_\ell^+$, we have $v\_coord_i \neq \bot$ in round r (line 18).
Proof of claim C. Let p_i be any process in X_ℓ^+. Let τ be the time at which p_i sets the timer at line 5 of round r. At this moment, since p_i finished executing line 4, there are at least $(n-t)$ processes from which p_i received an EA_PROP2$[r]()$ message. Since $|X_\ell^-| \geq t+1$, it follows that among these $(n-t)$ processes, there is at least one, say p_k, that belongs to X_ℓ^-. Observe that p_k necessarily broadcast the message EA_PROP2$[r]()$ before τ. Since $r > r_2$, this message is received by p_ℓ before time $\tau + \delta$.

Therefore, if p_ℓ did not broadcast a message EA_COORD$[r]()$ before receiving EA_PROP2$[r]()$ from p_k, as $p_k \in X_\ell^- \subseteq F(r)$, the condition of line 11 and the when statement preceding it are both satisfied, and p_ℓ broadcasts EA_COORD$[r]()$ at line 13. Consequently, in all cases, p_ℓ broadcasts a message EA_COORD$[r]()$ by time $\tau + \delta$. Finally, since $p_i \in X_\ell^+$ and $r > r_2$, this message is received by p_i before time $\tau + 2\delta$.

Let us recall that, as $r > r_1$, it holds that $r > 2\delta$, and consequently, since p_i set $timer_i[r]$ to r (line 8) at time τ, the timeout occurs after time $\tau + 2\delta$. Therefore, p_i receives the message EA_COORD$[r]()$ from p_ℓ before the timeout. Consequently, when evaluated by p_i, the predicate of line 17 is necessarily `false`, and $v\_coord_i \neq \bot$. This proves the claim.

We show in the following that all correct processes return the same value in round r. Let us first observe that every correct process broadcasts an EA_PROP2$[r]()$ message that carries a value which was necessarily ea-proposed by a correct process. Therefore, since p_ℓ is correct (and is the coordinator of r), the message EA_COORD$[r]()$ it broadcasts in round r contains a value, say w, that was sent to it by a correct process. Therefore, since (due to the definition of r), the processes of $F(r)$ are correct, the EA_RELAY$[r]()$ messages broadcast by them carry either w or $\bot$. Consequently, every correct process p_i can either returns w or val_i after executing the lines 7-10. To finish the proof, it remains to show that no correct process p_i returns val_i (if $val_i \neq w$).

Let us observe that each correct process waits at line 6 until it receives $(n-t)$ EA_RELAY$[r]()$ messages. Since $|X_\ell^+| > t$, it follows that at least one of these messages was broadcast by a process in X_ℓ^+. Due to Claim C, this message cannot carry $\bot$. It then follows from the predicate of line 7 that any correct process executes line 8 and returns w, which proves the lemma. $\Box_{Lemma\ 3}$

Theorem 3. *The algorithm described in Figure* 3 *implements an* m-valued eventual agreement *object in* $\mathcal{BZ\_AS}_{n,t}[t < n/3, \Diamond\langle t+1\rangle$*bisource*$]$.

Proof The proof follows from Lemma 1, Lemma 2, and Lemma 3. $\Box_{Theorem\ 3}$

5.4 Looking for efficiency: Parameterized eventual agreement

Time complexity of the EA algorithm The aim of the previous algorithm was to attain a round r during which all correct processes return the same value (ea-proposed by one of them). Hence its time complexity can be measured by the value of this round number. As the underlying synchrony assumption is *eventual*, we only know that this number r is finite.

Hence, to eliminate the noise created by the "eventual" attribute, and consequently be able to compute a time complexity of the algorithm, let us replace the $\Diamond\langle t+1\rangle$bisource synchrony assumption by the $\langle t+1\rangle$bisource assumption, i.e., we consider that there is a $\langle t+1\rangle$bisource from the very beginning. The corresponding system model is denoted $\mathcal{BZ\_AS}_{n,t}[t < n/3, \langle t+1\rangle\text{bisource}]$.

The uncertainty created by the "eventual" attribute is consequently eliminated, and the only uncertainty is the identity of the bisource and its associated input and output timely channels. As there are n processes and $\alpha = \binom{n}{n-t}$ combinations for the sets $F(r)$, it follows that the algorithm, which works in $\mathcal{BZ\_AS}_{n,t}[t < n/3, \Diamond\langle t+1\rangle\text{bisource}]$, terminates in at most αn rounds when the system behaves as $\mathcal{BZ\_AS}_{n,t}[t < n/3, \langle t+1\rangle\text{bisource}]$.

Improving the time complexity One way to improve the time complexity of the algorithm (as measured previously) is to consider a "tuning" parameter k, $0 \leq k \leq t$, and use it in both the synchrony assumption and the size of the sets $F(r)$, as follows.

- The assumption $\langle t+1 \rangle$bisource is replaced by the stronger assumption $\langle t+1+k \rangle$bisource.
- Instead of $(n-t)$, the size of the sets $F(r)$ is now $n-t+k$.

An algorithm, parameterized with k, extending the basic algorithm of Figure 3 and based on the previous definition is described in Appendix C. Designed for the system model $\mathcal{BZ\_AS}_{n,t}[t < n/3, \diamond\langle t+1+k \rangle \text{bisource}]$, this algorithm has a time complexity of βn where $\beta = \binom{n}{n-t+k}$ when executed in $\mathcal{BZ\_AS}_{n,t}[t < n/3, \langle t+1+k \rangle \text{bisource}]$.

As simple instances of this parameterized algorithm, let us consider two particular values of k. For $k = 0$, we obtain the basic algorithm. For $k = t$, the time complexity is n, which is the best that can be obtained with a round coordinator-based algorithm (up to n rounds can be needed to benefit from the $\langle t+1+k \rangle$bisource).

6 Byzantine Consensus Algorithm

m-Valued Byzantine consensus In the m-valued Byzantine consensus, the correct processes propose values from a set of at most m values. The corresponding object is a one-shot object, that provides the processes with a single operation denoted $\mathsf{CONS\_propose}(v)$, where v is the value proposed by the invoking process. This operation returns a value to the invoking process. If p_i obtains the value v, we say that it "decides" v. The consensus object is defined by the following properties.

- CONS-Termination. The invocation of $\mathsf{CONS\_propose}()$ by a correct process terminates.
- CONS-Validity. If a correct process decides a value v, there is a correct process that invoked $\mathsf{CONS\_propose}(v)$.
- CONS-Agreement. No two correct processes decide different values.

An algorithm solving m-valued Byzantine consensus Assuming $m \leq \lfloor \frac{n-(t+1)}{t} \rfloor$, the algorithm described in Figure 4 implements an m-valued consensus object in $\mathcal{BZ\_AS}_{n,t}[t < n/3, \langle t+1 \rangle \text{bisource}]$. This algorithm, which –thanks to the previous abstractions– is simple, uses the following underlying objects.

- Each process p_i manages a round number r_i (initialized to 0), and a current estimate of the decision value, denoted est_i.
- EA_OBJECT is an m-valued eventual-agreement object shared by all processes. Its aim is to allow the processes to eventually converge to the same estimate value. Hence, the associated line 4 is mainly related to the CONS-Termination property.
- $AC\_OBJECT[1..]$ is an unbounded array of m-valued adopt-commit objects, shared by all processes. $AC\_OBJECT[r]$ is the adopt-commit object used at round r. The aim of these objects (line 6) is to allow the correct processes to decide a value proposed by one of them, and to prevent them from deciding different values, i.e., to guarantee consensus safety.
- $CB[0]$ is a CB-broadcast instance, used at the very beginning to obtain a value proposed by a correct process and allow a process p_i to use the associated set $CB[0].cb\_valid_i$ to check the validity of the values returned by the EA_OBJECT object (i.e., to check if this value was proposed by a correct process).

When a correct process p_i invokes $\mathsf{CONS\_propose}(v_i)$, it first invokes $CB[0].\mathsf{CB\_broadcast}\textsc{valid}(v_i)$ to obtain a value for was proposed by a correct process (line 1)[4]. As already indicated, this invocation also ensures that the sets $CB[0].cb\_valid_i$ of correct processes are eventually equal and contain values proposed only by correct processes.

[4]Even if, up to now, a process behaved "correctly", it may crash in the future and become then faulty. Hence, no process can a priori considers the value it proposes as a value proposed by a correct process.

operation $\mathsf{CONS\_propose}(v_i)$ **is**
(1) $est_i \leftarrow CB[0].\mathsf{CB\_broadcast}\ \textsc{valid}(v_i)$; % safety: validity %
(2) **repeat forever**
(3) $r_i \leftarrow r_i + 1$;
(4) $v \leftarrow EA\_OBJECT.\mathsf{EA\_propose}(r_i, est_i)$; % liveness %
(5) **if** $(v \in CB[0].cb\_valid_i)$ **then** $est_i \leftarrow v$ **end if**; % safety: validity %
(6) $\langle tag, est_i\rangle \leftarrow AC\_OBJECT[r_i].\mathsf{AC\_propose}(est_i)$; % safety: agreement %
(7) **if** $(tag = \texttt{commit})$ **then** $\mathsf{RB\_broadcast}\ \textsc{decide}(est_i)$ **end if**
(8) **end repeat**.

when $\textsc{decide}(v)$ **is RB-delivered do**
(9) **if** ($\textsc{decide}(v)$ RB-delivered from $(t+1)$ diff. processes) **then** $\mathsf{return}(v)$ **end if**.

Figure 4: An algorithm for m-valued Byzantine consensus in $\mathcal{BZ\_AS}_{n,t}[t < n/3, \Diamond\langle t+1\rangle\text{bisource}]$

Then process p_i enters an infinite loop (lines 2-8). After it has entered its current round (line 3), process p_i proposes its current estimate of the decision value est_i to the to the EA object, namely, it invokes $EA\_OBJECT.\mathsf{ea\_propose}(r_i, est_i)$ (line 4). If the value returned by this invocation is a value that it knows as proposed by a correct process, it adopts it as new estimate, otherwise it keeps its previous estimate (line 5).

Process p_i proposes then the current value of est_i to the adopt-commit object associated with the current round, from which it obtains a pair $\langle tag, est_i\rangle$ (line 6). If the value of the tag is `commit` (line 7), p_i RB-broadcasts the message $\textsc{decide}(est_i)$ to inform the other processes that the value of est_i can be decided. Then, whatever the value of the tag, p_i proceeds to the next round with its (possibly new) estimate value est_i.

Finally, as soon as a process, that not yet decided, has RB-delivered the same message $\textsc{decide}(v)$ from $(t+1)$ different processes, it decides v and stops (line 9). Let us notice that at least one of these messages is from a correct process.

Theorem 4. *The algorithm described in Figure* 4 *solves the* m-valued Byzantine consensus *problem in the system model* $\mathcal{BZ\_AS}_{n,t}[t < n/3, \Diamond\langle t+1\rangle \textit{bisource}]$.

Proof We say that a process p_i starts round r when it assigns value r to its local variable r_i (line 3).
Proof of the CONS-Termination property.
If a process decides at line 9, it previously RB-delivered the message $\textsc{decide}(v)$ from $(t+1)$ different processes. Due to the RB-termination property of the corresponding $(t+1)$ RB-broadcasts, each correct process RB-delivers this message from the same set of $(t+1)$ processes, and consequently decides. So, let us assume by contradiction that no correct process decides at line 9.

Let us first observe that, due to the CB-Operation Termination property that no correct process p_i blocks forever at line 1. Moreover, it follows from the CB-Operation Validity that $CB[0].cb\_valid_i \neq \emptyset$ when this invocation terminates.

As no correct process decides, and all correct processes invoke $\mathsf{EA\_propose}(1, -)$, it follows from the EA-Termination and AC-Termination properties that they all terminate the first round, and consequently start the second. Moreover, if the estimate est_i of a correct process p_i is updated at line 5, its new value is a value proposed by a correct process. It follows that the correct processes start the second round with estimate values est_i containing values proposed by correct processes. As no correct process decides, the same reasoning applies to all rounds $r > 1$.

Let us observe that the local variables $CB[0].cb\_valid_i$ of the correct processes eventually converge to the same content (CB-Set Agreement and Termination properties of $CB[0]$). Hence, there is a round r_0 such that, for every correct process p_i, the set $CB[0].cb\_valid_i$ is never updated after it starts r_0.

It then follows from the EA-Eventual Agreement property of EA_OBJECT, that there is a round $r > r_0$ during which all correct processes obtain the same value v at line 4, where v is a value proposed by a correct process. Hence, since $r > r_0$, they all succeed the test of line 5 and adopt v as their new estimate est_i. Therefore, all correct processes invoke$AC_OBJECT[r].\mathsf{AC_propose}(v)$ at line 6. Due to the AC-Obligation property of $AC\_OBJECT[r]$, all correct processes obtain $\langle \texttt{commit}, v\rangle$ at line 6. Consequently, they all RB-broadcast the same message $\langle \texttt{commit}, v\rangle$ at line 7. An

$n - t \geq t + 1$, the decision predicate of line 9 becomes eventually true at every correct process, which contradicts the initial assumption.

Proof of the CONS-Validity property.
Let us consider the first round. Let p_i be a correct process. It follows from the CB-Operation Validity property of $CB[0]$ that est_i is a value proposed by a correct process. Moreover, it follows from the CB-Set Validity property, that $CB[0].cb\_valid_i$ contains only values proposed by correct processes. It follows from these observations that, be or not est_i modified at line 5, it contains a value proposed by a correct process when p_i invokes $AC\_OBJECT[1].\mathsf{AC\_propose}(est_i)$ at line 6. It then follows from the AC-Validity property of $AC\_OBJECT[1]$ that the value assigned to est_i at line 6 is a value proposed by a correct process. The same reasoning applies iteratively to all rounds, from which it follows that a value that is RB-broadcast by a correct process at line 7 is a value proposed by a correct process.

If a correct process p_i decides a value v at line 9, it follows from the decision predicate used at this line that v was RB-broadcast at line 7 by at least one correct process p_j. The previous paragraph has shown that such a value v was proposed by a correct process.

Proof of the CONS-Agreement property.
Let us first observe that, if a correct process decides at line 9, it decides a value RB-broadcast by a correct process at line 7. Hence, the proof consists in showing that no two correct processes RB-broadcast different values at line 7.

Let r be the first round at which a correct process p_i RB-broadcast a message DECIDE() at line 7. Let v the value carried by this message. It follows that, at line 6, p_i obtained the pair $\langle \texttt{commit}, v\rangle$ from the object $AC\_OBJECT[r]$. Let us consider another correct process p_j. There are two cases.

- p_j RB-broadcast the message DECIDE(w) at line 9 of round r. This means that it obtained $\langle \texttt{commit}, w\rangle$ from $AC\_OBJECT[r]$. It then follows from the AC-agreement property of the object $AC\_OBJECT[r]$ that $v = w$. Moreover, p_j proceeds to the next round with $est_j = v$.
- p_j did not RB-broadcast the message DECIDE(w) at line 9 of round r. It then follows from the AC-agreement property of $AC\_OBJECT[r]$ that p_j obtained the pair $\langle \texttt{adopt}, v\rangle$. Hence, at line 6, p_j assigned the value v to est_j.

It follows that the estimate values of all the correct processes that progress to the next round are equal to v. Let p_x be any correct process executing round $(r + 1)$. It follows from the EA-Validity property of EA_OBJECT, that the invocation by p_x of $EA\_OBJECT.\mathsf{EA\_propose}(r + 1, est_x)$ returns v, and from the AC-Obligation property of $AC\_OBJECT[r + 1]$ that this object returns $\langle -, v\rangle$ to p_x. This means that the estimates of all the correct processes remain forever equal to v. Hence, no value different from v can be RB-broadcast at line 7 by a correct process during a round $r' \geq r$. $\Box_{Theorem\ 4}$

7 Conclusion

A variant To ensure that a value decided by a correct process is always a value that was proposed by a correct process, this paper has considered m-valued consensus, i.e., there is a value that is proposed by at least m correct processes. To ensure that no value proposed only by Byzantine processes is ever decided, some Byzantine consensus algorithms (e.g., [10, 23]) do not have such an "m-valued" requirement. They instead allow the correct processes to decide a default value $\perp$ when they do not propose the same value. The algorithms proposed in the paper can be modified to satisfy this different validity requirement.

The aim and the content of the paper This paper presented a consensus algorithm for asynchronous Byzantine message-passing systems, that is optimal with respect to the underlying synchrony assumption. This assumption is the existence of a process that is an *eventual* $\langle t + 1\rangle$*bisource*. Such a process p is a non-faulty process that eventually has

(a) timely input channels from t correct processes and (b) timely output channels to t correct processes. Moreover these input and output channels can connect p to different subsets of processes.

In addition to a reliable broadcast abstraction, the design of the algorithm, which is very modular, is based on simple abstractions: a new broadcast abstraction called *cooperative broadcast*, adopt-commit objects that cope with Byzantine processes (hence, as far as we know, the paper presented the first implementation of such objects in the presence of Byzantine processes), and a new round-based object called *eventual agreement*, whose definition involves a pretty weak validity property.

This paper answered a long-lasting problem, namely, solving Byzantine consensus with the weakest underlying synchrony assumptions. Finally, as claimed in the introduction, and in addition to its optimality with respect to synchrony requirements, a very important first class property of the proposed algorithm lies in its *design simplicity*. "Simplicity $\Rightarrow$ easy" is rarely true for non-trivial problems [2].

Acknowledgments

This work has been partially supported by the French ANR project DISPLEXITY devoted to computability and complexity in distributed computing, and the Franco-German ANR project DISCMAT devoted to connections between mathematics and distributed computing.

References

[1] Aguilera M.K., Delporte-Gallet C., Fauconnier H., and Toueg S., Consensus with Byzantine failures and little system synchrony. *Proc. 45th IEEE/IFIP Int'l Conference on Dependable Systems and Networks (DSN'06)*, IEEE Press, pp. 147-155, 2006.

[2] Aigner M. and Ziegler G., *Proofs from THE BOOK* (4th edition). Springer, 274 pages, 2010 (ISBN 978-3-642-00856-6).

[3] Attiya H. and Welch J., Distributed computing: fundamentals, simulations and advanced topics, (2d Edition), *Wiley-Interscience*, 414 pages, 2004.

[4] Baldellon O., Mostéfaoui A. and Raynal M., A necessary and sufficient synchrony condition for solving Byzantine consensus in symmetric networks. *Proc. 12th Int'l Conference on Distributed Computing and Networks (ICDCN'11)*, Springer LNCS 6522, pp. 215-226, 2011.

[5] Ben-Or M., Another advantage of free choice: completely asynchronous agreement protocols. *Proc. 2nd Annual ACM Symposium on Principles of Distributed Computing(PODC'83)*, ACM Press, pp. 27-30, 1983.

[6] Bracha G., Asynchronous Byzantine agreement protocols. *Information & Computation*, 75(2):130-143, 1987.

[7] Bracha G. and Toueg S., Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824-840, 1985.

[8] Cachin Ch., Kursawe K., and Shoup V., Random oracles in Constantinople: practical asynchronous Byzantine agreement using cryptography. *Proc. 19th Annual ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 123-132, 2000.

[9] Chandra T. and Toueg S., Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225-267, 1996.

[10] Correia M., Ferreira Neves N., and Verissimo P., From consensus to atomic broadcast: time-free Byzantine-resistant protocols without signatures. *The Computer Journal*, 49(1):82-96, 2006.

[11] Delporte-Gallet C., Devismes S., Fauconnier H. and Larrea M., Algorithms for extracting timeliness graphs. *17th Int'l Colloquium on Structural Information and Communication Complexity (SIROCCO'10)*, Springer LNCS 6058, pp. 127-141, 2010.

[12] Doudou A., Garbinato B., Guerraoui R. and Schiper A., Muteness failure detectors: specification and implementation. *3rd European Dependable Computing Conference (EDCC'99)*, Springer LNCS 1667, pp. 71-87, 1999.

[13] Dwork C., Lynch N., and Stockmeyer L., Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2), 288-323, 1988.

[14] Fischer M.J., Lynch N.A., and Paterson M.S., Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382, 1985.

[15] Friedman R., Mostéfaoui A., and Raynal M., Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46-56, 2005.

[16] Gafni E., Round-by-round fault detectors: unifying synchrony and asynchrony. *Proc. 17th ACM Symposium on Principles of Distributed Computing (PODC)*, ACM Press, pp. 143-152, 1998.

[17] Herlihy M.P., Kozlov D., and Rajsbaum S., *Distributed computing through combinatorial topology*, Morgan Kaufmann/Elsevier, 336 pages, 2014 (ISBN 9780124045781).

[18] Kihlstrom K.P., Moser L.E. and Melliar-Smith P.M., Byzantine fault detectors for solving consensus. *The Computer Journal*, 46(1):16-35, 2003.

[19] Lamport L., Shostack R., and Pease M., The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3)-382-401, 1982.

[20] Lynch N.A., *Distributed algorithms*. Morgan Kaufmann Pub., San Francisco (CA), 872 pages, 1996 (ISBN 1-55860-384-4).

[21] Mostéfaoui A., Moumen H., and Raynal M., Signature-free asynchronous Byzantine consensus with $t < n/3$ and $O(n^2)$ messages. *Proc. 33th ACM Symposium on Principles of Distributed Computing (PODC'14)*, ACM Press, pp. 2-9, 2014.

[22] Mostéfaoui A. and Raynal M., Solving consensus using Chandra-Toueg's unreliable failure detectors: a general quorum-based approach. *Proc. 13th Int'l Symposium on Distributed Computing (DISC'99)*, Springer, LNCS #1693, pp. 49âĂŞ63, 1999.

[23] Mostéfaoui A. and Raynal M., Signature-free broadcast-based intrusion tolerance: never decide a Byzantine value. *Proc. 14th Int'l Conference On Principles Of Distributed Systems (OPODIS'10)*, Springer LNCS 6490, pp. 144-159, 2010.

[24] Moumen H., Mostéfaoui A., and Trédan G., Byzantine consensus with few synchronous links. *Proc. 11th Int'l Conference On Principles Of Distributed Systems (OPODIS'07)*, Springer LNCS 4878, pp. 76-89, 2007.

[25] Pease M., R. Shostak R., and Lamport L., Reaching agreement in the presence of faults. *Journal of the ACM*, 27:228-234, 1980.

[26] Rabin M., Randomized Byzantine generals. *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS'83)*, IEEE Computer Society Press, pp. 116-124, 1983.

[27] Raynal M., *Fault-tolerant agreement in synchronous message-passing systems*. Morgan & Claypool, 165 pages, 2010 (ISBN 978-1-60845-525-6).

A On the Definition of $\diamond\langle t+1\rangle$bisource

Our definition of an $\diamond\langle t+1\rangle$bisource is slightly different from the original definition introduced in [1]. The difference is that it considers only eventually timely channels connecting correct processes, while [1] considers eventually timely channels connecting a correct process to correct or faulty processes. Hence, an $\diamond\langle t+1\rangle$bisource is an $\diamond\langle 2t+1\rangle$bisource in the parlance of [1]. We consider only eventually timely channels connecting pair of correct processes for the following reason: an eventually timely channel connecting a correct process and a Byzantine process can always appear to the correct process as being an asynchronous channel.

B An algorithm implementing RB-broadcast

The distributed algorithm described in Figure 5 implements the RB-broadcast abstraction. It is a simplified version of an algorithm due to G. Bracha [6]. Its proof can be found in [6].

When a process p_i wants to RB-broadcast a message whose content is v_i, it broadcasts the message $\text{INIT}(i, v_i)$ (line 1). When a process p_i receives a message $\text{INIT}(j, -)$ for the first time, it broadcasts a message $\text{ECHO}(j, v)$ where v is the data content of the $\text{INIT}()$ message (line 2). If the message $\text{INIT}(j, v)$ received is not the first message $\text{INIT}(j, -)$, p_j is Byzantine and the message is discarded. Finally, when p_i has received the same message $\text{ECHO}(j, v)$ from more than $(n+t)/2$ processes, it broadcasts the message $\text{READY}(i, v_i)$ (line 4). The fact that $\text{ECHO}(j, v)$ was received from more than $(n+t)/2$ processes ensures that no two correct processes can broadcast different messages $\text{READY}(j, -)$, but it is still possible that correct processes broadcast such a message while other correct processes never broadcast a message $\text{READY}(i, -)$.

The aim of lines 6-8) is to prevent deadlock. Finally, the aim of lines 9-11 is to ensure that all or none of the non-faulty processes RB-deliver the message $\text{MSG}(j, v)$ from p_j. To this end, the RB-delivery predicate requires that p_i receives $(2t+1)$ copies of $\text{READY}(j, v)$, which means at least $(t+1)$ copies from non-faulty processes (line 9).

operation RB_broadcast MSG(v_i) **is**
broadcast INIT(i, v_i).

when INIT(j, v) **is received do**
if (first reception of INIT$(j, -)$) **then** broadcast ECHO(j, v) **end if**.

when ECHO(j, v) **is received do**
if (ECHO(j, v) received from more than $\frac{n+t}{2}$ different processes and READY(j, v) not yet broadcast)
then broadcast READY(j, v)
end if.

when READY(j, v) **is received do**
if (READY(j, v) received from $(t+1)$ different processes and READY(j, v) not yet broadcast)
then broadcast READY(j, v)
end if;
if (READY(j, v) received from $(2t+1)$ different processes and MSG(j, v) not yet RB-delivered)
then RB_deliver $(j$,MSG$(v))$ % RB-delivery of the message MSG(v) from p_j %
end if.

Figure 5: Implementing RB-broadcast in $\mathcal{BZ\_AS}_{n,t}[t < n/3]$ (from [6])

C Parameterized Eventual Agreement Object

Let k such that $0 \leq k \leq t$. The algorithm described in Figure 6 implements an EA object in the system model $\mathcal{BZ\_AS}_{n,t}[t < n/3, \Diamond\langle t+1+k\rangle\text{bisource}]$. As already indicated in Section 5.4, it also assumes that each set $F(r)$ contains $(n-t+k)$ processes.

This algorithm is a simple extension of the basic EA algorithm of Figure 3. The lines with the same numbers are the exactly the same in both algorithms. The lines N1, N2, and N3, are new lines, and the lines 12.M1, 12.M2, 13.M1, and 13.M2, replace the lines 12 and 13 of Figure 3. Hence, the modifications are restricted to lines N1-N3 and lines 12.M1-13.M2.

- The first modification (lines N1-N3) is as follows. Before setting the timer of the current round (line 5), the processes executes an additional communication phase: each process p_i broadcasts the message EA_PROP3$[r_i](bag_i)$ where bag_i contains the $(n-t)$ messages EA_PROP2$[r_i]()$ that made true the predicate of line 3. Then, p_i waits until it received a message EA_PROP3$[r_i]()$ from $(n-t)$ processes. This wait is to not set the timer too early, and (as in Figure 3) makes its expiration dependent only of the message EA_COORD$(r)()$ sent by the round coordinator at line 13.M1, and the messages EA_RELAY$(r)()$ sent by the processes of $F(r)$ at line 18.
- As now $|F(r)| = n-t+k$, it is possible that $F(r)$ always contain k Byzantine processes. Consequently, the round cordinator waits until it obtains the same message EA_PROP2$[r_i](w)$ (contained in a message EA_PROP3$[r]()$, as defined at line N1) from $(k+1)$ processes in $F(r)$ (lines 12.M1-12.M2). As at least one of these processes is correct, it follows that EA_PROP2$[r](w)$ is from a correct process. Hence, when the $\langle t+1+k\rangle$bisource coordinates a round, it selects and broadcasts (line 13.M1) a value w obtained from a correct process.

Remark When we consider the basic algorithm of Figure 3, in addition to a CB-broadcast, an invocation of EA_propose() involves three sequential communication steps (messages EA_PROP2$[r]()$, EA_COORD$[r]()$, and EA_REALY$[r]()$). Differently, the algorithm of Figure 6 requires an additional communication step (messages EA_PROP3$[r]()$).

operation EA_propose$_k(r_i, val_i)$ **is** % $0 \le k \le t$ %
(1) $aux_i \leftarrow CB[r_i]$.CB_broadcast EA_PROP1(val_i);
(2) broadcast EA_PROP2$[r_i](aux_i)$;
(3) wait (EA_PROP2$[r_i]()$ messages have been received from $(n-t)$
different processes, and their aux values belong to $CB[r_i].cb\_valid_i$);
(4) **if** (the $(n-t)$ previous messages contain the same value v) **then** return(v) **end if**;
(N1) **let** bag_i **be** the bag of the $(n-t)$ previous messages;
(N2) broadcast EA_PROP3$[r_i](bag_i)$;
(N3) wait (EA_PROP3$[r_i]()$ messages have been received from $(n-t)$ diff. processes);
(5) set $timer_i[r_i]$ to r_i;
(6) wait (EA_RELAY$[r_i](aux)$ messages received from $(n-t)$ different processes);
(7) **if** (EA_RELAY$[r_i](v)$ where $v \neq \perp$ received from a process in $F(r_i)$)
(8) **then** return(v)
(9) **else** return(val_i)
(10) **end if**.

when EA_PROP3$[r]()$ **is received from** a process in $F(r)$ **do**
(11) **if** $((i = \text{coord}(r)$ $\wedge$ (EA_COORD$[r]()$ not already broadcast))
(12.M1) **then if** ($\exists\, w$: EA_PROP2$[r](w)$ belong to the bag carried by each message
(12.M2) EA_PROP3$[r]()$ received from $(k+1)$ processes in $F(r)$)
(13.M1) **then** broadcast EA_COORD$[r](w)$
(13.M2) **end if**
(14) **end if**.

when EA_COORD$[r](v)$ **is received from** $p_{\text{coord}(r)}$ or ($timer_i[r]$ expires) **do**
(15) **if** (EA_RELAY$[r]()$ not already broadcast)
(16) disable $timer_i[r]$;
(17) **if** ($timer_i[r]$ expired) **then** $v\_coord_i \leftarrow \perp$ **else** $v\_coord_i \leftarrow v$ **end if**;
(18) broadcast EA_RELAY$[r](v_coord_i)$
(19) **end if**.

Figure 6: An algorithm for an m-valued EA object in $\mathcal{BZ\_AS}_{n,t}[t < n/3, \diamond\langle t+1+k\rangle \text{bisource}]$